ICML 2026

V_1: Unifying Generation and Self-Verification for Parallel Reasoners

Harman Singh, Xiuyu Li, Kusha Sareen, Monishwaran Maheswaran, Sijun Tan, Xiaoxia Wu, Junxiong Wang, Alpay Ariyak, Qingyang Wu, Samir Khaki, Rishabh Tiwari, Long Lian, Yucheng Lu, Boyi Li, Alane Suhr, Ben Athiwaratkun, Kurt Keutzer

UC Berkeley · Mila · Together AI · NVIDIA

Harman Singh

PhD Student, UC Berkeley · LLMs, Reasoning & Reinforcement Learning

arXiv:2603.04304

github.com/HarmanDotpy/pairwise-self-verification

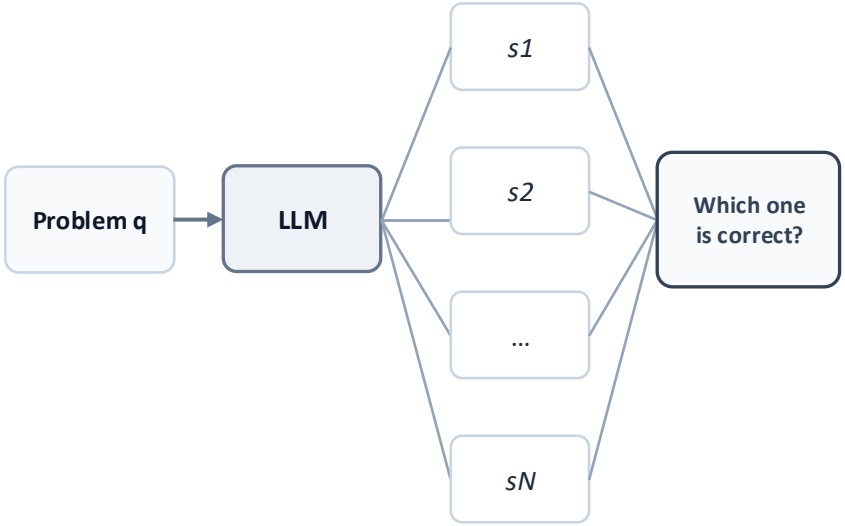
Parallel Reasoning Has a Verification Bottleneck

MOTIVATION

Parallel test-time scaling: sample N independent chains of thought, then aggregate. Pass@N grows quickly — the correct solution is usually in the set.

Aggregation is the hard part: majority voting needs objective, exactly-matchable answers (math). Hard to make it work for code, software agents, or open-ended domains.

Self-verification: the same model must identify the correct candidate — no external reward model, no test execution, no oracle at inference time.



Sampling N is only as useful as your ability to pick the winner. On hard LiveCodeBench-v6 problems, Pass@1 is 40.2% — accurate selection recovers 63.9%, closing most of the gap to Pass@N.

Why Current Approaches Fail

EMPIRICAL ANALYSIS — SECTION 3

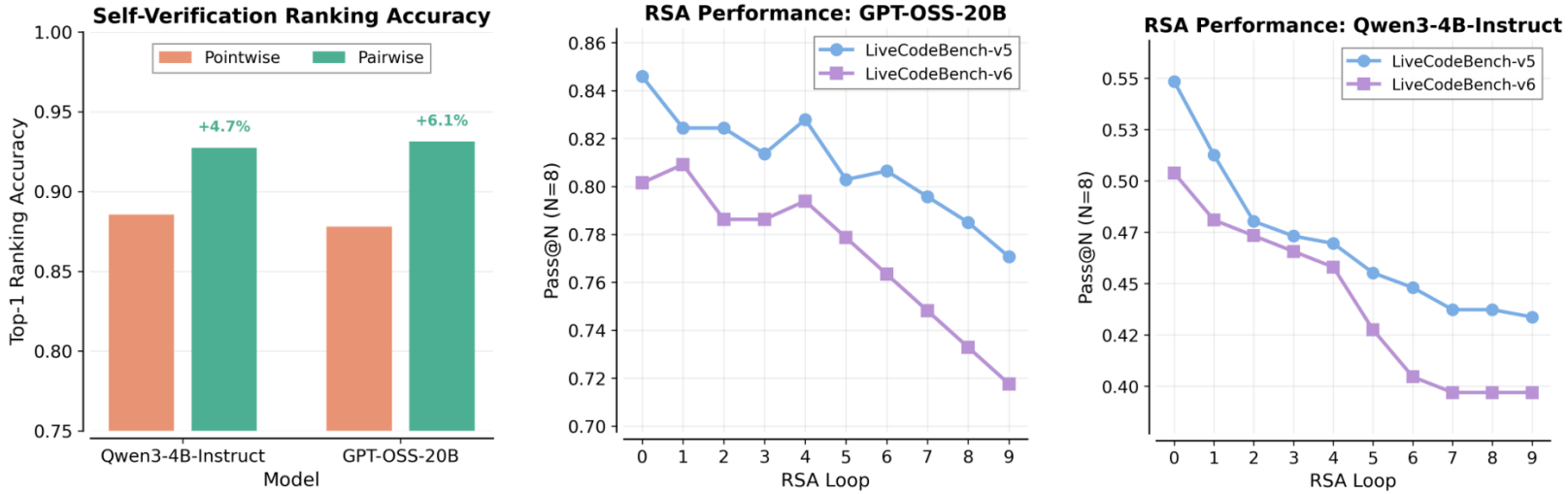


Figure 1 — (left) pairwise vs pointwise top-1 ranking accuracy; (middle, right) Pass@N under recursive self-aggregation (RSA).

Pointwise scoring → calibration collapse. Latent utilities are identifiable only up to monotonic transforms (Bradley–Terry); scores saturate — e.g., 12/16 candidates rated 10/10.

Self-aggregation (RSA) → diversity collapse. Pass@N decreases monotonically with aggregation steps — correct outlier solutions are discarded during refinement.

The V_1 Framework

CONTRIBUTIONS

Core insight: relative comparison is a fundamentally more robust verification primitive than absolute scoring — exploit it at inference and during RL training.

V_1-Infer inference-time, training-free

- Uncertainty-guided pairwise ranking via a Swiss-system tournament
- Allocates verification compute to the most uncertain candidate pairs
- O(N)-scale comparisons instead of C(N,2) — near-Pass@N selection
- **Up to +10% Pass@1 over pointwise verification; beats RSA at a fraction of the calls**

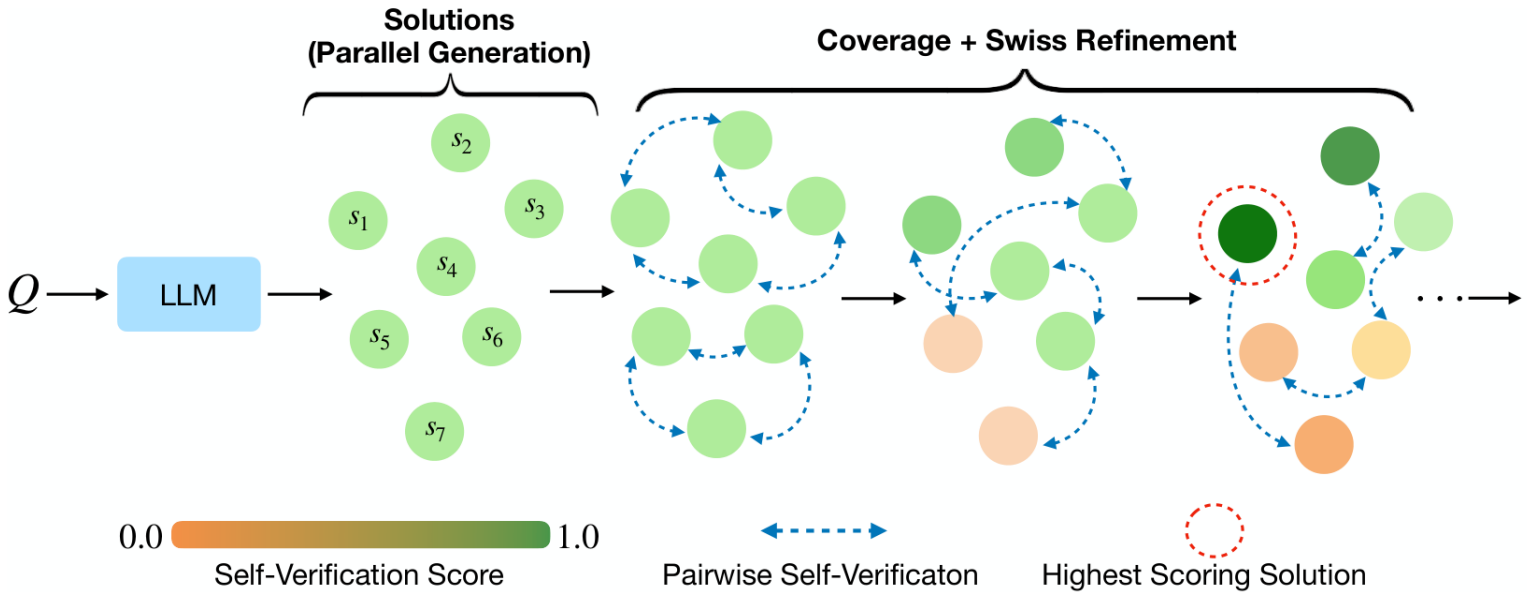
V_1-PairRL RL post-training

- One model co-trained as generator and pairwise self-verifier
- Online, co-evolving objective — verifier always sees in-distribution rollouts
- Reward design that prevents two collapse modes of joint training
- **+7–9% test-time scaling over pointwise co-training; +8.7% base Pass@1 over standard RL**

Evaluated on code generation (LiveCodeBench-v5/v6, CodeContests, SWE-bench Lite) and math reasoning (AIME'25, HMMT'25) across GPT-OSS-20B/120B, Qwen3-4B-Instruct/Thinking, and Gemini-2.5-Flash.

V_1-Infer: Tournament-Style Pairwise Self-Verification

METHOD — SECTION 4



Solutions are pairwise self-verified in rounds; scores sharpen and comparisons concentrate where ranking is still uncertain.

Naive pairwise ranking needs C(N,2) judge calls — quadratic. **V_1-Infer reaches near-Pass@N selection with an O(N)-scale budget** by treating pair selection as an active-learning problem.

Uncertainty-Weighted Score Aggregation

METHOD — JUDGING AND SCORING

Rated comparisons, not binary verdicts. Each judge call scores both solutions of a pair on a calibrated 1–10 scale — a decisive 9-vs-3 carries more evidence than a marginal 6-vs-5.

Rating gap as a confidence proxy. The weight w_{ij} scales each outcome by the judge's margin; floor $\tau = 0.1$ keeps near-ties informative but weak.

Score = weighted win rate. μ_i aggregates over the opponent set N(i): confident judgments dominate the ranking, ambiguous ones contribute little variance.

Pair outcome and ratings

$$v_{ij} \in \{0,\ 0.5,\ 1\} \quad \text{(loss / tie / win)}, \quad (r_i, r_j) \in [1, 10]$$

Confidence weight

$$w_{ij} = \max\left(\frac{|r_i - r_j|}{9},\ \tau\right)$$

Estimated quality score (Eq. 1)

$$\mu_i = \frac{\sum_{j \in \mathcal{N}(i)} w_{ij} v_{ij}}{\sum_{j \in \mathcal{N}(i)} w_{ij}}$$

Same 1–10 scale is used for the pointwise baseline — performance gaps isolate the comparison structure, not the scoring format.

Two-Phase Budget Allocation

METHOD — ALGORITHM 1

Phase 1 — Topology Coverage

Random disjoint pairs guarantee global connectivity, then every solution is raised to a minimum degree $d_{min} = 2$ by pairing low-degree nodes with their closest-μ peers — anchoring prevents orphaned, path-dependent rankings.

Phase 2 — Swiss Refinement

Sort by μ; pair unseen near-neighbours within a window. Under Bradley–Terry, near-ties yield the highest marginal information gain — an active-learning rule that spends the remaining budget exactly at the ambiguous decision boundaries.

Algorithm 1 Uncertainty-Guided Pairwise Ranking

Require: Problem x, candidates $\mathcal{S} = \{s_i\}_{i=1}^{N}$, pairwise budget B, min-degree $d_{\min}$, Swiss window size h, weight floor τ
Ensure: Ranking π

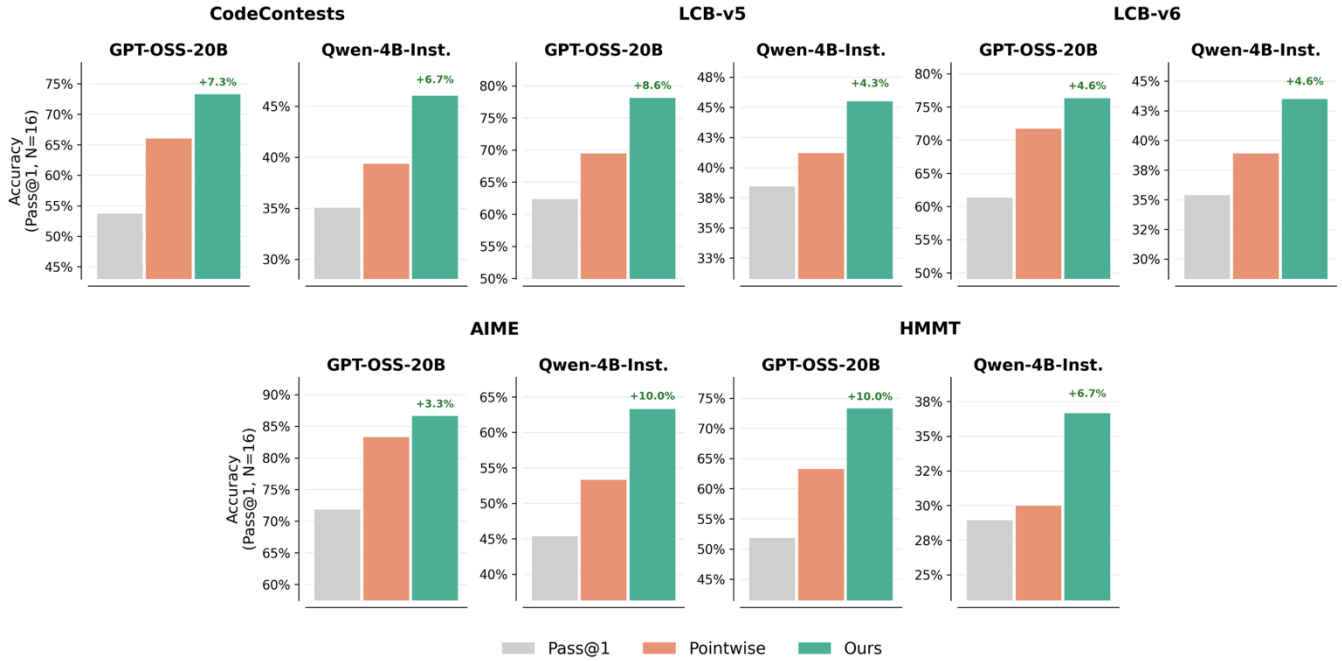
Initialize: $\forall\, i$, scores $\mu_i \leftarrow 0.5$ and degrees $d_i \leftarrow 0$
  history $\mathcal{H} \leftarrow \emptyset$ ▷ $(i,j) \in \mathcal{H}$ iff (s_i, s_j) has been compared
  state $\mathcal{T} \leftarrow (\{\mu_i\}_{i=1}^{N}, \{d_i\}_{i=1}^{N}, \mathcal{H})$
used $\leftarrow 0$
Phase 1: Topology Coverage
while used $< B$ **and** $\exists i : d_i < d_{\min}$ **do**
  $\mathcal{P} \leftarrow$ CoveragePairs$(\mathcal{T}, d_{\min})$
  $\mathcal{O} \leftarrow$ PairSelfVerify$(x, \mathcal{S}, \mathcal{P})$ ▷ parallel LLM judging
  UpdateStats$(\mathcal{T}, \mathcal{O}, \tau)$ ▷ update μ_i, d_i; record $\mathcal{H}$
  used $\leftarrow$ used $+|\mathcal{P}|$
end while
Phase 2: Swiss Refinement
while used $< B$ **and** $N > 2$ **do**
  $\pi \leftarrow$ Rank$(\{\mu_i\}_{i=1}^{N})$ ▷ descending μ
  $\mathcal{P} \leftarrow$ SwissPairs$(\pi, \mathcal{T}, h)$ ▷ within window h; prefer unseen and near-ties
  $\mathcal{O} \leftarrow$ PairSelfVerify$(x, \mathcal{S}, \mathcal{P})$
  UpdateStats$(\mathcal{T}, \mathcal{O}, \tau)$
  used $\leftarrow$ used $+|\mathcal{P}|$
end while
return Rank$(\{\mu_i\}_{i=1}^{N})$

Budget B set as 1×, 2×, or 3× the N base generations; judge calls within a round run in parallel.

No per-task tuning: $d_{min} = 2$, $h = 8$, $\tau = 0.1$ are fixed across every benchmark and model in the paper.

V_1-Infer Consistently Beats Pointwise Self-Verification

RESULTS — N = 16 (Fixing base number of solutions and applying pairwise/pointwise verification on it)



+10.0%
HMMT, GPT-OSS-20B — largest math gain

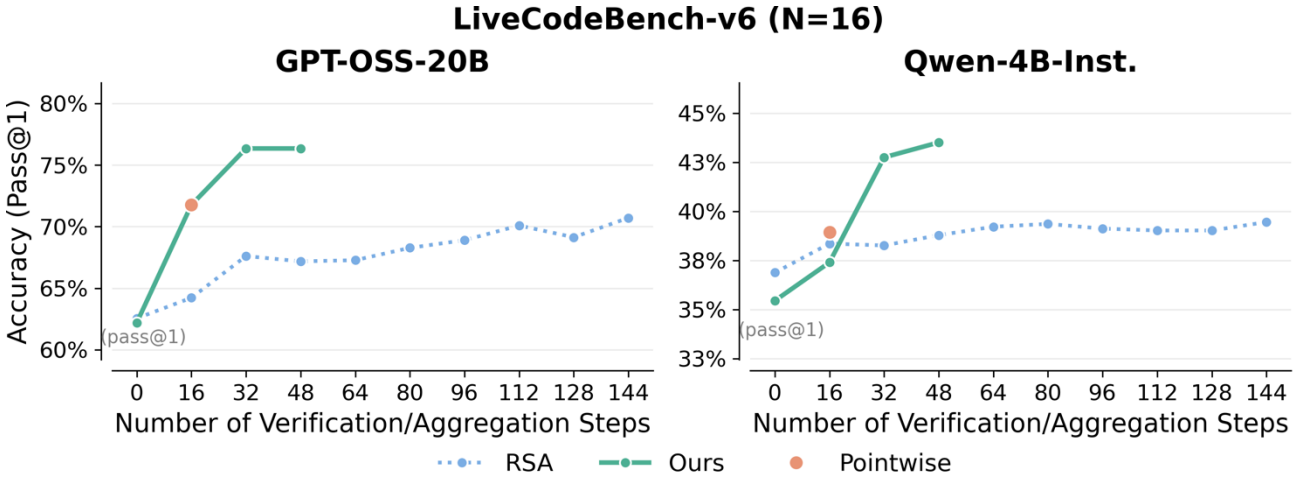
+8.6%
LiveCodeBench-v5, GPT-OSS-20B

+7.3%
CodeContests: 66.1% → 73.3%

Gains hold across all four models (GPT-OSS-20B/120B, Qwen3-4B-Instruct/Thinking), five benchmarks, and grow or stay stable as verification budget increases (1× → 3×).

Better Test-Time Scaling Than Aggregation

RESULTS — BUDGET-MATCHED COMPARISON WITH RSA



Recursive Self-Aggregation run with population N = 16, aggregation size k = 4, T = 10 steps.

76%

Pass@1 on LCB-v6 with only 48 judge calls — above RSA's best at 144+ calls

33% → 45%

pointwise vs pairwise at an equal 64-call budget (Qwen3-4B, code avg.)

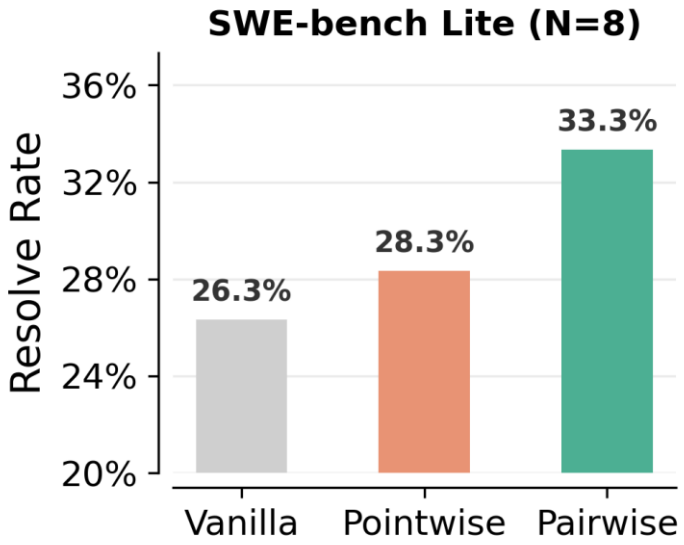
Pass@N kept

selection never destroys diversity — unlike aggregation

Can be used in an orthogonal manner as well: used as the fitness signal inside RSA's evolutionary loop, pairwise selection reaches 93.3% on AIME'25 with lower latency than vanilla RSA.

Generalizes to Agentic Software Engineering

RESULTS — SWE-BENCH LITE, 300 GITHUB ISSUES



Resolve rate over N = 8 candidate patches.

Setup. Gemini-2.5-Flash generates 8 candidate patches per issue via an agentic pipeline (mini-swe-agent); the same model then self-verifies to pick one.

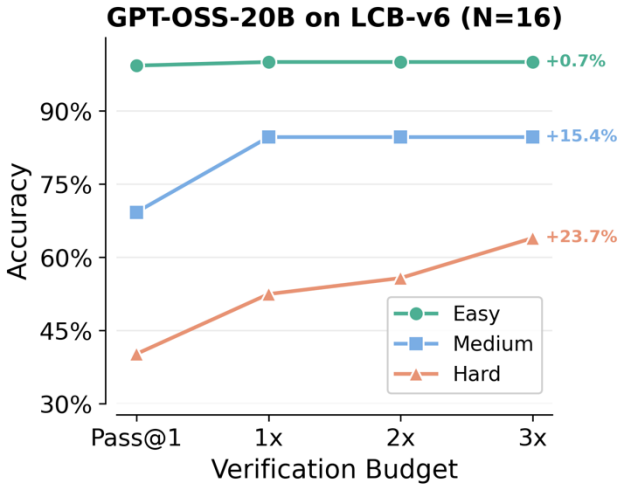
Pure self-verification. The judge sees only the issue text and candidate diffs — no repository context, no execution feedback, no agent trajectory.

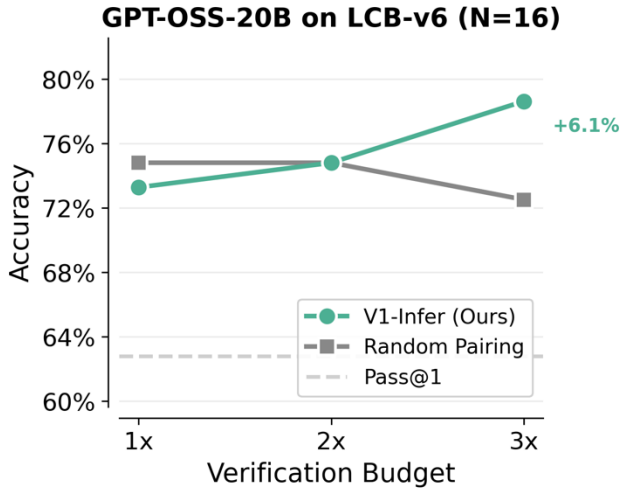
Result. Pairwise 33.3% vs pointwise 28.3% vs first-candidate 26.3% — head-to-head diff comparison surfaces root-cause fixes over surface-level patches (e.g., a fix in the parent DraggableBase class vs the same idea applied only to one subclass).

Why it matters for agents: at deployment there is no reward signal — ranking your own trajectories or patches is the only scalable selection mechanism.

Where the Gains Come From

ANALYSIS & ABLATIONS





GPT-OSS-20B on LiveCodeBench-v6, N = 16.

Largest gains where selection matters most. Hard problems: 40.2% → 63.9% (+23.7); medium +15.4; easy already at the 99.3% ceiling.

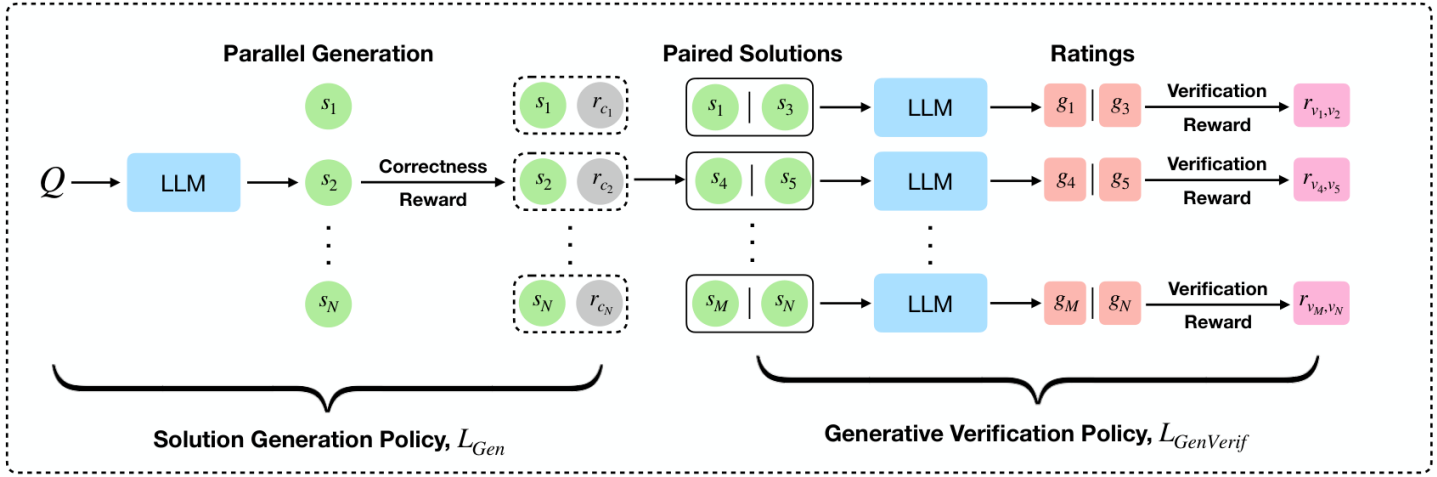
Pair selection is not free. Uncertainty-guided pairing beats random pairing at every matched budget — +6.1% at 3× here (+3.8 in the paper's 3× ablation).

Interpretation. Pairwise verification converts the Pass@1 → Pass@N gap into realized accuracy precisely on the diverse, ambiguous candidate sets.

Qualitative failure mode of pointwise: score saturation — e.g., 15/16 candidates rated 10/10, making selection effectively random among ties.

V_1-PairRL: Co-Evolving Solver–Verifier Training

METHOD — SECTION 5



$$J(\theta) = J_{\text{Gen}}(\theta) + \lambda J_{\text{PairVerif}}(\theta)$$

Single policy, two objectives, shared rollouts. Each step, the model generates G solutions per problem (GRPO group); the same rollouts provide correctness rewards for J_{Gen} and ground-truth-labelled pairs for $J_{\text{PairVerif}}$. As the generator improves, the verifier trains on the shifting, in-distribution solution distribution — eliminating the train/inference mismatch of offline or external verifiers.

Reward Design: Making Joint Training Stable

REWARDS and REWARD HACKING

Generation: binary r ∈ {0,1} — all ground-truth tests pass (0 on malformed output). **Verification:** judge scores each solution of a pair, normalized to v ∈ [0,1], rewarded against true correctness y:

$$r_{\text{verif}} = \frac{1}{2} \sum_{i \in \{A,B\}} \mathbb{1}(|v_i - y_i| \le 0.2) \cdot (1 - |v_i - y_i|)$$

Collapse 1: Safe verifier output collapse

Without the sparsity threshold, the verifier emits v ≈ 0.5 for everything: never very wrong, never informative.

Fix: reward only when |v − y| ≤ 0.2 — score correct ≥ 0.8 or incorrect ≤ 0.2. Forces committed judgments.

Collapse 2: Empty-Solution Loop

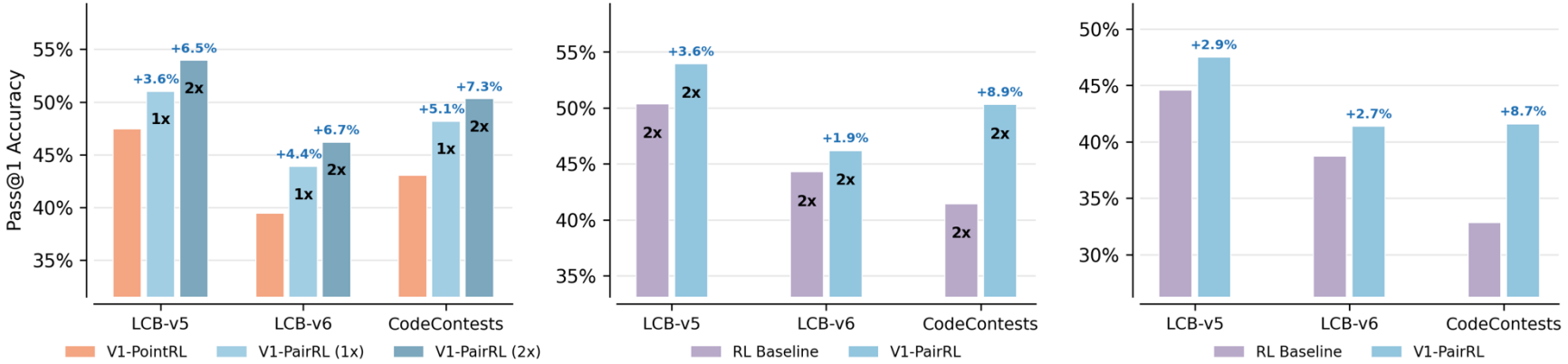
Incorrect–incorrect pairs are trivially judged, so the generator degrades toward easy-to-reject garbage while the verifier farms reward.

Fix: only form pairs containing at least one correct solution (Correct–Incorrect or Correct–Correct).

Both interventions are necessary: removing the sparsity threshold collapses solution-generation performance, not just verification.

V_1-PairRL: Stronger Scaling and a Stronger Generator

RESULTS — QWEN3-4B-INSTRUCT TRAINED ON DEEPCODER, N = 16



(left) test-time scaling vs pointwise co-training · (middle) vs the RL baseline, both using V1-Infer at 2× budget · (right) base Pass@1, no test-time scaling

+6.5 / +6.7 / +7.3

over pointwise co-training (V1-PointRL) on LCB-v5 / v6 / CodeContests — the pairwise objective, not co-training alone, drives the gain

+3.6 / +1.9 / +8.9

over the standard-RL baseline when both use identical V1-Infer (2×) at inference — training created capability, not just compatibility

+2.9 / +2.7 / +8.7

base Pass@1 with no test-time scaling — co-training improves the generator itself

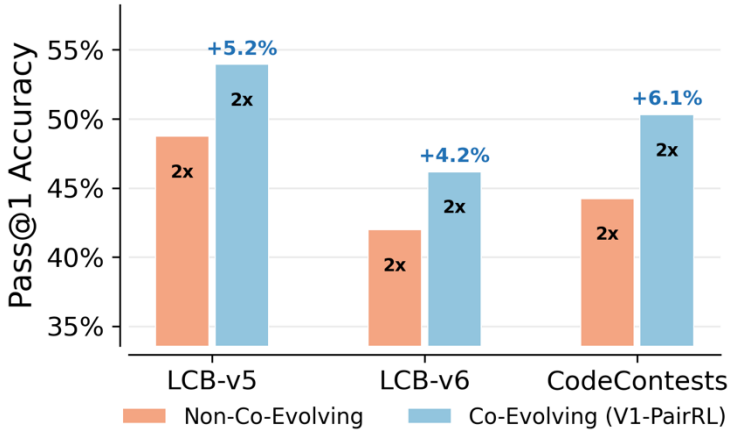
Co-Evolution Is the Critical Ingredient

ABLATION & TRAINING SETUP

Ablation. A non-co-evolving variant trains the same joint objective, but on verification pairs generated offline from the base model. It loses on every benchmark — **the verifier must track the policy's moving solution distribution.**

Training configuration

- Qwen3-4B-Instruct-2507 on DeepCoder (24K verified coding problems), 150 steps
- GRPO config: clip-high 0.28, token-level loss, no KL; no std normalization
- Compute-matched: 8 rollouts/problem — 4 solver + 4 verifier vs 8 solver for the baseline (longer baseline training does not close the gap)
- λ = 1.0; verification advantage = REINFORCE with a mean baseline across prompts



Both checkpoints evaluated with V1-Infer at 2× budget.

TAKEAWAYS

1

Pairwise comparison is the right self-verification primitive

Well-posed relative judgments avoid the calibration collapse of absolute scoring and the diversity collapse of aggregation.

2

Verification compute can be allocated intelligently

Uncertainty-guided Swiss refinement reaches near-Pass@N selection at O(N) cost — up to +10% over pointwise, ahead of RSA at a fraction of the calls, and it transfers to agentic SWE-bench.

3

Generation and verification should co-evolve in RL

One policy, shared rollouts, careful anti-hacking rewards: +7–9% test-time scaling and up to +8.7% base Pass@1 over standard RLVR.

Where this goes next: pairwise verification over agent trajectories and tool-use traces; process-level comparison signals during RL; multimodal verifiers; and verification-aware budget allocation as a learned policy.

Thank you · arXiv:2603.04304 · github.com/HarmanDotpy/pairwise-self-verification